Group Members:

Ismael Barajas - ismaelbarajas30@csu.fullerton.edu

Patrick Mahoney - patrick.mahoney@csu.fullerton.edu

Zach Sarvas - zsarvas@csu.fullerton.edu

About:

This project creates a Twitter-like microblogging service where users can post messages, repost messages, view global messages, see a timeline of messages from other users, narrow timelines to consist of only users they are following, follow or unfollow users, update user profile info, see who you are following and who is following you, like messages, create polls, and view/respond to polls. Asynchronous messaging queues are used for Asynchronous Posting and performing background processing using workers. This service is implemented using 5 RESTful back-end services which make use of SQL databases, Beanstalkd, Redis, DynamoDB, http basic authentication, and a service registry.

How To Run:

- Please make sure you are running the latest version of Redis/Hiredis, some features used are not compatible with older versions of Redis.
- Set up DynamoDB locally
- Navigate to your DynamoDB server folder and run this command to start DynamoDB server:

```
java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
```

- Run ./bin/dynamoDB_init.py from the command line to create tables for the Dynamo DataBase

- Replace your haproxy.cfg file with the provided file found in the “etc” folder called “haproxy.cfg”.
- Start Debugging SMTP server in a terminal by running this command:

```
python -m smtpd -n -c DebuggingServer localhost:1025
```

- Run ./bin/init.sh from the command line to create SQL databases.
- Run ./bin/foremanStart.sh from the command line to start foreman.
- Run sudo systemctl restart haproxy.

After following the above steps, the five microservices are now running as users.localhost, timelines.localhost, likes.localhost, and polls.localhost respectively. Registry is the only microservice not connected to the haproxy load balancer: http://localhost:5000.

API Documentation:

- ## User Microservice (URL: users.localhost):
    - **Connection:** close
    - **content-type:** application/json; charset=utf-8
    - **server:** gunicorn/20.0.4
  - **authenticated(auth_user):**
    - **Description:** This API is for validating the user login. If the username and password match the values in the database, we return the value “true”. If the validation returns false, we return 401 Status code which is “Unauthorized error”.
    - **Uses basic authentication**
    - **Endpoint**: /user/authenticate
    - **Method**: GET
    - **Parameters**: **username** (text), **password** (text)
    - **Sample Request:**

```
http -a ProfAvery:password users.localhost/user/authenticate
```

    - **Success Response:**

```
HTTP/1.1 200 OK
{
        "bio": "wow does this work",
        "email": "kavery@fullerton.edu",
        "id": 1,
        "password": "password",
        "username": "ProfAvery"
}
```

    - **Error Response:**

```
HTTP/1.1 401 Unauthorized
{
        "errors": {
                "Invalid Authentication": "Provided Basic HTTP
                Authentication credentials were invalid"
        }
}
```

  - **user(username):**

- **Description:** Takes one parameter **username**, where **username** is the desired account information. GETs the account information for the given **username**.
- **Endpoint**: /user/<username>
- **Method**: GET
- **Parameters**: **username** (text)
- **Sample Request:**

```
http GET users.localhost/user/ProfAvery
```

- **Success Response:**

```
HTTP/1.1 200 OK
{
        "bio": "wow does this work",
        "email": "kavery@fullerton.edu",
        "id": 1,
        "password": "password",
        "username": "ProfAvery"
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "ProfAver does not exist",
        "status": "404 Not Found"
}
```

- **update_user(user_id, column, content):**
  - **Description:** Takes three parameters **user_id**, **column**, and **content**, where **user_id** is the desired account, the **column** is the desired information to change, and **content** is the updated content. PUTs the updated account information into the table and removes the old content.
  - **Endpoint**: /user/update
  - **Method**: PUT
  - **Parameters**: **user_id** (number), **column** (text [username, email, password, bio]), **content** (text)
  - **Sample Request:**

```
http PUT users.localhost/user/update/ user_id=2 column=username
```

```
content=newUsername
```

- **Success Response:**

```
HTTP/1.1 200 OK
{
        "message": "Your username has been updated to newUsername"
}
```

- **Error Response:**

```
HTTP/1.1 400 Bad Request
{
        "message": "The one or all of the inputs you provided is invalid, check
        documentation for correctness.",
        "status": "400 Bad Request"
}
```

- **delete_user(user_id):**
  - **Description:** Takes one parameter **user_id**, where **user_id** is the desired account. DELETEs the account of the given **user_id**.
  - **Endpoint**: /user/delete
  - **Method**: DELETE
  - **Parameters**: **user_id** (number)
  - **Sample Request:**

```
http DELETE users.localhost/user/delete user_id=1
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
        "message": "User has been deleted."
}
```

  - **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "User you are trying to delete doesn't exist.",
        "status": "404 Not Found"
}
```

- **new_user(username, email, password, bio):**
  - **Description:** Takes 4 parameters **username**, **email**, **password**, and **bio**, where **username**, **email**, **password**, and **bio** are the information for the new user. POSTs a new account with the given information.
  - **Endpoint**: /user/new
  - **Method**: POST
  - **Parameters**: **username** (text), **email** (text), **password** (text), **bio** (text)
  - **Sample Request:**

```
http POST users.localhost/user/new/ username=Test
email=testemail@gmail.com password=test bio='wow so cool'
```

  - **Success Response:**

```
HTTP/1.1 201 Created
{
        "bio": "wow so cool",
        "email": "testemail@gmail.com",
        "id": 5,
        "password": "test",
        "username": "Test"
}
```

  - **Error Response:**

```
HTTP/1.1 409 Conflict
{
        "message": "UNIQUE constraint failed: users.email",
        "status": "409 Conflict"
}
```

- **followers(username):**
  - **Description:** Takes one parameter **username**, where **username** is the desired account. GETs the account information of the followers of the specified account.
  - **Endpoint**: /user/<username>/followers
  - **Method**: GET
  - **Parameters**: **username** (text)

- **Sample Request:**

```
http GET users.localhost/user/ProfAvery/followers
```

- **Success Response:**

```
HTTP/1.1 200 OK
{
        "follower_user": "KevinAWortman"
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "ProfAver=y does not exist",
        "status": "404 Not Found"
}
```

- **following(username):**
  - **Description:** Takes one parameter **username**, where **username** is the desired account. GETs the account information of who the specified account is following.
  - **Endpoint**: /user/<username>/following
  - **Method**: GET
  - **Parameters**: **username** (text)
  - **Sample Request:** http GET users.localhost/user/<username>/following

```
http GET users.localhost/user/ProfAvery/following
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
        "following_user": "KevinAWortman"
        },
        {
        "following_user": "Beth_CSUF"
        }
```

  - **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "ProfAveASry does not exist",
        "status": "404 Not Found"
}
```

- **follow(follower_username, following_username):**
  - **Description:** Takes two parameters **follower_username**, **following_username**, where **follower_username** is the account that wants to follow another, and **following_username** is the account that is to be followed. POSTs the new connection between these accounts to the followers table.
  - **Endpoint**: /user/follow
  - **Method**: POST
  - **Parameters**: **follower_username** (text), **following_username** (text)
  - **Sample Request:**

```
http POST users.localhost/user/follow follower_username=TestUser
following_username=ProfAvery
```

  - **Success Response:**

```
HTTP/1.1 201 Created
{
        "follower_id": 4,
        "following_id": 1,
        "id": 6
}
```

  - **Example Error Response:**

```
HTTP/1.1 409 Conflict
{
        "message": "You already follow ProfAvery",
        "status": "409 Conflict"
}
```

- **unfollow(follower_username, following_username):**

- **Description:** Takes two parameters **follower_username**, **following_username**, where **follower_username** is the account that wants to follow another, and **following_username** is the account that is to be followed. DELETEs the connection between these accounts from the followers table.
- **Endpoint**: /user/unFollow
- **Method**: DELETE
- **Parameters**: **follower_username** (text), **following_username** (text)
- **Sample Request:**

```
http DELETE users.localhost/user/unFollow follower_username=TestUser
following_username=ProfAvery
```

- **Success Response:**

```
HTTP/1.1 201 Created
{
        "message": "TestUser has un-followed ProfAvery"
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
  "message": "User you are trying to unFollow does not exist",
  "status": "404 Not Found"
}
```

- # Timeline Microservice:
    - **Connection:** close
    - **Server:** gunicorn/20.0.4
    - **content-type:** application/json; charset=utf-8
  - **public_timeline():**
    - **Description:** Takes no parameters. GETs the posts of all users and lists them in descending chronological order by timestamp.
    - **Endpoint**: /posts/all
    - **Method**: GET
    - **Parameters**: **username** (text)
    - **Sample Request:**

```
http GET timelines.localhost/posts/all
```

    - **Success Response:**

```
HTTP/1.1 200 OK
[
        {
        "id": 7,
        "original_post_url": "http://timelines.localhost/posts/6",
        "repost": 1,
        "text": "#cpsc315 #engr190w NeurIPS is $25 for students and $100 for
non-students this year!
https://medium.com/@NeurIPSConf/neurips-registration-opens-soon-6711158
1de99",
        "timestamp": "2021-10-23 04:52:13",
        "username": "TestUser"
        },
        {
        "id": 6,
        "original_post_url": null,
        "repost": 0,
        "text": "#cpsc315 #engr190w NeurIPS is $25 for students and $100 for
non-students this year!
https://medium.com/@NeurIPSConf/neurips-registration-opens-soon-6711158
1de99",
        "timestamp": "2021-10-23 04:52:13",
        "username": "Beth_CSUF"
        },
        {
        "id": 5,
```

```
        "original_post_url": null,
        "repost": 0,
        "text": "I keep seeing video from before COVID, of people not
needing to mask or distance, and doing something like waiting in line at
Burger King. YOU'RE WASTING IT!",
        "timestamp": "2021-10-23 04:52:12",
        "username": "KevinAWortman"
        },
        {
        "id": 4,
        "original_post_url": null,
        "repost": 0,
        "text": "If academia were a video game, then a 2.5 hour administrative
meeting that votes to extend time 15 minutes is a fatality. FINISH HIM",
        "timestamp": "2021-10-23 04:52:12",
        "username": "KevinAWortman"
        },
        {
        "id": 3,
        "original_post_url": null,
        "repost": 0,
        "text": "Yes, the header file ends in .h. C++ is for masochists.",
        "timestamp": "2021-10-23 04:52:12",
        "username": "ProfAvery"
        },
        {
        "id": 2,
        "original_post_url": null,
        "repost": 0,
        "text": "FYI: https://www.levels.fyi/still-hiring/",
        "timestamp": "2021-10-23 04:52:12",
        "username": "ProfAvery"
        },
        {
        "id": 1,
        "original_post_url": null,
        "repost": 0,
        "text": "Meanwhile, at the R1 institution down the street...
https://uci.edu/coronavirus/messages/200710-sanitizer-recall.php",
        "timestamp": "2021-10-23 04:52:12",
        "username": "ProfAvery"
        }
]
```


- **Error Response:**

```
NO ERROR RESPONSE
```

- **user_public_timeline(username):**
  - **Description:** Takes one parameter **username**, where **username** is the desired account. GETs the posts a user with **username** has posted and lists them in descending chronological order by timestamp.
  - **Endpoint**: /<username>/posts
  - **Method**: GET
  - **Parameters**: **username** (text)
  - **Sample Request:**

```
http GET timelines.localhost/ProfAvery/posts
```

  - **Success Response:**

```
HTTP/1.1 200 OK
[
        {
        "id": 1,
        "original_post_url": null,
        "repost": 0,
        "text": "Meanwhile, at the R1 institution down the street...
https://uci.edu/coronavirus/messages/200710-sanitizer-recall.php",
        "timestamp": "2021-10-23 04:52:12",
        "username": "ProfAvery"
        },
        {
        "id": 2,
        "original_post_url": null,
        "repost": 0,
        "text": "FYI: https://www.levels.fyi/still-hiring/",
        "timestamp": "2021-10-23 04:52:12",
        "username": "ProfAvery"
        },
        {
        "id": 3,
        "original_post_url": null,
        "repost": 0,
        "text": "Yes, the header file ends in .h. C++ is for masochists.",
        "timestamp": "2021-10-23 04:52:12",
        "username": "ProfAvery"
        }
]
```

  - **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "ProfAvey does not exist.",
```

```
        "status": "404 Not Found"
}
```

- **user_home_timeline(auth_user, username)**
  - **Description:** Takes two parameters **auth_user** and **username**, where **auth_user** is the authorized user and **username** is the entered username. The authorized user is checked with "username". GETs the posts by all users that this user follows and lists them in descending chronological order by timestamp. Uses Hug Basic Authentication to validate the user.
  - **Uses basic authentication: username** (text), **password** (text)
  - **Endpoint**: /<username>/home
  - **Method**: GET
  - **Parameters**: username (text), basic authentication
  - **Sample Request:**

```
http -a ProfAvery:password timelines.localhost/ProfAvery/home
```

  - **Success Response:**

```
HTTP/1.1 200 OK
[
        {
        "id": 6,
        "original_post_url": null,
        "repost": 0,
        "text": "#cpsc315 #engr190w NeurIPS is $25 for students and $100 for
non-students this year!
https://medium.com/@NeurIPSConf/neurips-registration-opens-soon-6711158
1de99",
        "timestamp": "2021-10-23 04:52:13",
        "username": "Beth_CSUF"
        },
        {
        "id": 4,
        "original_post_url": null,
        "repost": 0,
        "text": "If academia were a video game, then a 2.5 hour administrative
meeting that votes to extend time 15 minutes is a fatality. FINISH HIM",
        "timestamp": "2021-10-23 04:52:12",
        "username": "KevinAWortman"
        },
        {
        "id": 5,
```

```
        "original_post_url": null,
        "repost": 0,
        "text": "I keep seeing video from before COVID, of people not needing to mask or distance, and doing something like waiting in line at Burger King. YOU'RE WASTING IT!",
        "timestamp": "2021-10-23 04:52:12",
        "username": "KevinAWortman"
        }
]
```


- **Error Response:**


```
HTTP/1.1 401 Unauthorized
{
        "errors": {
        "Invalid Authentication": "Provided Basic HTTP Authentication credentials were invalid"
        }
}
```


- **get_post_by_id(id):**
  - **Description:** Takes one parameter **id**, where **id** is the user id of the desired account. GETs the posts of a user with the desired user ID and lists them in descending chronological order by timestamp.
  - **Endpoint**: /posts/id
  - **Method**: GET
  - **Parameters**: **id** (number)
  - **Sample Request:**

```
http GET timelines.localhost/posts/1
```

  - **Success Response:**


```
HTTP/1.1 200 OK
{
        "id": 1,
        "original_post_url": null,
        "repost": 0,
        "text": "Meanwhile, at the R1 institution down the street... https://uci.edu/coronavirus/messages/200710-sanitizer-recall.php",
        "timestamp": "2021-10-23 04:52:12",
        "username": "ProfAvery"
}
```


  - **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "post_id:234 does not exist",
        "status": "404 Not Found"
}
```

- **new_post(auth_user, text):**
  - **Description:** Takes two parameters **auth_user** and **text**, where **auth_user** is the authorized user and **text** is the entered username. POSTs a new user with the desired username.
  - **Uses basic authentication:** username, password
  - **Endpoint**: /compose/post
  - **Method**: POST
  - **Parameters**: **text** (text), basic authentication
  - **Sample Request:**

```
http -a ProfAvery:password POST http://timelines.localhost/compose/post
text="this is a test post boi"
```

  - **Success Response:**

```
HTTP/1.1 201 Created
{
        "id": 8,
        "text": "this is a test post boi",
        "username": "ProfAvery"
}
```

  - **Error Response:**

```
HTTP/1.1 400 Bad Request
{
        "errors": {
        "text": "Required parameter 'text' not supplied"
        }
}
```

- **async_new_post(auth_user, txt):**
  - **Description:** Takes **2** parameters **auth_user** and **txt**, where **auth_user** is the authorized user and **txt** is the text desired for the user's post. POSTs a new post with the desired **txt** from the given **auth_user** using a worker queue.
  - **Endpoint**:/compose/async/post

- **Method**: POST
- **Parameters**: **auth_user** (text), **txt** (text)
- **Sample Request:**

```
http -a ProfAvery:password POST
http://timelines.localhost/compose/async/post text="something
important is being said here"
```

- **Success Response:**

```
HTTP/1.0 202 Accepted
{
   "message": "Your post has been created."
}
```

- **Error Response:**

```
HTTP/1.1 400 Bad Request
server: gunicorn/20.0.4
{
        "errors": {
                "text": "Required parameter 'text' not supplied"
        }
}
```

- **delete_post(id):**
  - **Description:** Takes one parameter **post_id** which is the desired id of the post being deleted. DELETEs the post of the given **post_id**.
  - **Endpoint**:/delete/post/{id}
  - **Method**: DELETE
  - **Parameters**: **id** (number)
  - **Sample Request:**

```
http DELETE timelines.localhost/delete/post/1
```

  - **Success Response:**

```
HTTP/1.1 200 OK
```

```
{
        "message": "Post:1 has been deleted."
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "Post:1 does not exist.",
        "status": "404 Not Found"
}
```

- **new_repost(id, auth_user):**
  - **Description:** Takes two parameters **id** and **auth_user**, where **id** is the entered user ID and **auth_user** is the authorized user. POSTs a new user with the desired username.
  - **Endpoint**: /repost/<id>
  - **Method**: POST
  - **Parameters**: **auth_user** (text), **text** (text)
  - **Sample Request:**

```
http -a ProfAvery:password POST timelines.localhost/repost/1
```

  - **Success Response:**

```
HTTP/1.1 201 Created
{
        "id": 9,
        "original_post_url": "http://timelines.localhost/posts/1",
        "repost": true,
        "text": "Meanwhile, at the R1 institution down the street...
https://uci.edu/coronavirus/messages/200710-sanitizer-recall.php",
        "username": "ProfAvery"
}
```

  - **Error Response:**

```
HTTP/1.1 409 Conflict
{
        "message": "You have already reposted this post",
        "status": "409 Conflict"
}
```

• Like Microservices:

- **Connection:** close
- **Server:** gunicorn/20.0.4
- **content-type:** application/json; charset=utf-8

- **like(post_id, username):**
  - **Description:** Takes **2** parameters **post_id** and **username**, where **username** is the user “liking” a post, and **post_id** is the post the user is “liking”. POSTs a “like” under the account **username** for the post with the specified **post_id**. Asynchronous message is added to a message queue to verify that the post being liked exists, if the post does not exist the post is unliked and an email is sent to the user notifying them that the post does not exist.
  - **Endpoint**: /like/
  - **Method**: POST
  - **Parameters**: **username** (text), **post_id** (number)
  - **Sample Request:**

```
http POST likes.localhost/like username='ProfAvery' post_id=1
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
        "message": "ProfAvery has liked /posts/1"
}
```

  - **Error Response:**

```
HTTP/1.1 403 Forbidden
{
        "message": "ProfAvery has already like /posts/1 ",
        "status": "403 Forbidden"
}
```

- **unlike(post_id, username):**
  - **Description:** Takes **2** parameters **post_id** and **username**, where **username** is the user “unliking” a post, and **post_id** is the post the user is “unliking”. DELETEs a “like” from the post with the specified **post_id** using the account **username**.
  - **Endpoint**: /unlike/

- **Method**: DELETE
- **Parameters**: **username** (text), **post_id** (number)
- **Sample Success Request:**

```
http DELETE likes.localhost/unlike username='ProfAvery' post_id=1
```

- **Success Response:**

```
HTTP/1.1 200 OK
{
        "message": "ProfAvery has unliked /posts/1"
}
```

- **Sample Error Request:**

```
http DELETE likes.localhost/unlike username='KevinAWortman'
post_id=5
```

- **Error Response:**

```
HTTP/1.1 403 Forbidden
{
        "message": "KevinAWortman has not liked /posts/5",
        "status": "403 Forbidden"
}
```

- **post_likes(post_id):**
  - **Description:** Takes **1** parameter **post_id** which is the ID of the post. GETs the total likes for the post with the specified **post_id**.
  - **Endpoint**: /like/<post_id>
  - **Method**: GET
  - **Parameters**: **post_id** (number)
  - **Sample Request:**

```
http GET likes.localhost/like/1
```

- **Success Response:**

```
HTTP/1.1 200 OK
{
        "likes": 1,
        "post_id": "/posts/1"
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "/posts/8 does not exist.",
        "status": "404 Not Found"
}
```

- **user_likes(username):**
  - **Description:** Takes **1** parameter **username**, and checks if the user exists. GETs the liked posts of the user with **username**.
  - **Endpoint**: /like/posts/<username>
  - **Method**: GET
  - **Parameters**: **username** (text)
  - **Sample Request:**

```
http GET likes.localhost/like/posts/ProfAvery
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
        "liked_posts":
        [
                "/posts/1"
```

```
        ],
        "username": "ProfAvery"
}
```

- **Error Response:**

```
HTTP/1.1 404 Not Found
{
        "message": "ProfAver does not exist.",
        "status": "404 Not Found"
}
```

- **popular_posts( ):**
  - **Description:** Takes **no** parameters. GETs the top 5 posts (out of all posts) with the greatest amount of likes.
  - **Endpoint**: /like/popular
  - **Method**: GET
  - **Parameters**: NA
  - **Sample Request:**

```
http GET likes.localhost/like/popular
```

  - **Success Response:**

```
HTTP/1.1 200 OK
{
        "popular_posts": [
                {
                        "/posts/1": 2
                },
                {
                        "/posts/2": 1
                }
```

```
        ]
}
```

- **Error Response: NA**

Poll Microservices:

create_poll(username, question, options):

- **Description:** Takes **3** parameters **username, question,** and **options**, where **username** is the user posting a poll, **question** is the question other users will answer, and the **options** are the choices other users can select (**options**: 2-4). POSTs a poll with the given **question** and **options**, with the assigned **poll_id** if the **username** does not already exist in **users_voted**.
- **Endpoint**: /poll/
- **Method**: POST
- **Parameters**: **username** (text)**, question** (text)**, options** (multiple/array)
- **Sample Request:**

```
http POST polls.localhost/poll username='ProfAvery' question='test question' options:='["is that a plane? Nope just an option", "There goes another option", "Another option", "this is an option"]'
```

- **Success Response:**

```
HTTP/1.0 200 OK
{
  "message": "ProfAvery has created a Poll",
  "poll": {
    "details": {
      "options": [
        {
          "option_1": "is that a plane? Nope just an option",
          "votes": 0
        },
        {
          "option_2": "There goes another option",
          "votes": 0
        },
        {
          "option_3": "Another option",
          "votes": 0
        },
        {
```

```
                "option_4": "this is an option",
                "votes": 0
            }
        ],
        "question": "test question",
        "users_voted": {}
    },
    "poll_id": "1",
    "username": "ProfAvery"
  }
}
```


- **Error Response:**


```
HTTP/1.0 400 Bad Request
{
  "message": "Please provide at least 2 to 4 poll options.",
  "status": "400 Bad Request"
}
```


- **results_poll (username, poll_id):**
  - **Description:** Takes 2 parameters **username** and **poll_id**, where **username** is the name of the **user**, and **poll_id** is the poll ID of that user's **poll**. GETs the poll by the **username** and **poll_id**.
  - **Endpoint**: /poll/results/<username>/<poll_id>
  - **Method**: GET
  - **Parameters**: **username** (text), **poll_id (**number**)**
  - **Sample Request:**

```
http GET polls.localhost/poll/results/ProfAvery/1
```

  - **Success Response:**


```
HTTP/1.0 200 OK
{
  "details": {
    "options": [
      {
        "option_1": "is that a plane? Nope just an option",
```

```
            "votes": "0"
        },
        {
            "option_2": "There goes another option",
            "votes": "0"
        },
        {
            "option_3": "Another option",
            "votes": "0"
        },
        {
            "option_4": "this is an option",
            "votes": "0"
        }
    ],
    "question": "test question",
    "users_voted": {}
  },
  "poll_id": "1",
  "username": "ProfAvery"
}
```


- **Error Response:**


```
HTTP/1.0 404 Not Found

{
  "message": "The poll you are looking for does not exist.",
  "status": "404 Not Found"
}
```


- **update_poll(poll_id, owner_username, voter_username, vote):**
  - **Description:** Takes 4 parameters **poll_id**, **owner_username, voter_username,** and **vote**. Where **poll_id** is the ID of the poll being voted on, **owner_username**

is the owner of the poll, **voter_username** is the user voting in the poll, and **vote** is the number option **voter_username** is choosing (option: 1-4). PUTs the **vote** . # include what is being stored for keeping track of vote count/user

- **Endpoint**: /poll/vote/
- **Method**: PUT
- **Parameters**: **poll_id** (number)**, owner_username** (text)**, voter_username** (text)**, vote** (number)
- **Sample Request:**

```
http PUT polls.localhost/poll/vote poll_id='1'
owner_username='ProfAvery' voter_username='TestUser' vote=3
```

- **Success Response:**

```
HTTP/1.0 200 OK
{
   "message": "Your vote for option 3 has been taken."
}
```

- **Error Response:**

```
HTTP/1.0 403 Forbidden

{
   "message": "TestUser has already voted in ProfAvery's poll.",
   "status": "403 Forbidden"
}
```

- **delete_poll(username, poll_id):**
  - **Description:** Takes 2 parameters **username** and **poll_id**, where **username** is the user who owns the poll, and **poll_id** is the ID of the poll being deleted. DELETEs the poll with the given **username** and **poll_id**.
  - **Endpoint**: /poll/delete
  - **Method**: DELETE
  - **Parameters**: **username** (text), **poll_id** (number)

- **Sample Request:**

```
http DELETE polls.localhost/poll/delete poll_id='1'
username='ProfAvery'
```

- **Success Response:**

```
HTTP/1.0 200 OK
{
   "message": "ProfAvery has successfully deleted poll_id:1."
}
```

- **Error Response:**

```
HTTP/1.0 404 Not Found
{
   "message": "Poll you are trying to delete does not exist.",
   "status": "404 Not Found"
}
```

• Service Registry Microservices (localhost:5000):

○ register(service, url):

- **Description:** Takes 2 parameters **service** and **url**, where **service** is the name of the service (e.g. 'users', 'posts', or 'likes'), and **url** is the url of the service being registered. POSTs the **url** for the **service**.
- **Endpoint**: /register
- **Method**: POST
- **Parameters**: **service** (text), **url** (text)
- **Sample Request:**

```
http POST localhost:5000/register service='timelines'
url='http://timelines.localhost '
```

- **Success Response:**

```
HTTP/1.0 200 OK
```

```
{
   "likes": {
      "http://127.0.1.1:5300": true
   },
   "polls": {
      "http://127.0.1.1:5400": true
   },
   "timelines": {
      "http://127.0.1.1:5200": true,
      "http://127.0.1.1:5201": true,
      "http://127.0.1.1:5202": true,
      "http://timelines.localhost": true
   },
   "users": {
      "http://127.0.1.1:5100": true
   }
}
```


- **Error Response:**


```
HTTP/1.0 422 Unprocessable Entity
{
   "message": "timelinesa does not exist.",
   "status": "422 Unprocessable Entity"
}
```


- **avaliable_services(service):**
  - **Description:** Takes 1 parameter **service**, which is the name of the service. GETs the availability of the service in the database.
  - **Endpoint**: /available/<service>
  - **Method**: GET
  - **Parameters**: **service** (text)
  - **Sample Request:**

```
http GET localhost:5000/available/timelines
```

- **Success Response:**

```
HTTP/1.0 200 OK
{
   "timelines": {
      "http://127.0.1.1:5200": true,
      "http://127.0.1.1:5201": true,
      "http://127.0.1.1:5202": true
   }
}
```

- **Error Response:**

```
HTTP/1.0 404 Not Found
{
   "message": "timelineasd does not exist.",
   "status": "404 Not Found"
}
```

- **health_check():**
  - On startup, registry starts a daemon thread targeting the health_check() function which runs health checks on the self registering services every 20 seconds. When a request fails the service url that failed gets removed from the available services dictionary.

```
registry = {
   "users": {},
   "timelines": {},
   "likes": {},
   "polls": {}
}

# Runs health checks on services
def health_check():
   _lock = threading.Lock()
   while 1:
      for api in registry:
         for url in copy.deepcopy(registry[api]):
            try:
```

```
                r = requests.get(f"{url}/health-check")
                r.raise_for_status()
            except requests.HTTPError:
                with _lock:
                    del registry[api][url]
        time.sleep(20)

# Starts daemon thread that runs forever, that checks health of all services
@hug.startup()
def run_heath_check(api):
    threading.Thread(target=health_check, daemon=True).start()
```

- **Workers:**
    - **Timelines Worker:**
        - Retrieves the job from the “timeline” tube queue then processes the message.
        - Then inserts a new post into the database.
        - Once a post is added to the database the worker creates a new job in the “poll” tube to verify if a post that contains a link to a poll does, in fact, refer to a valid poll.
    - **Likes Worker:**
        - Retrieves the job from the “like” tube queue then processes the message.
        - First checks the posts database to verify if the post being liked exists, if so nothing happens.
        - If the post does not exist then the like is removed from the Redis datastore.
        - Then the worker creates a new job in the “email” tube to send an email notifying the user that the post has been unliked.
    - **Polls Worker:**
        - Retrieves the job from the “poll” tube queue then processes the message.
        - A regular expression is used to find any poll links in the text of the post.
        - If a poll link is found then a request to that url is made to verify that the poll exists.
        - If the poll does not exist then the post is deleted from the Posts database.

        - Then the worker creates a new job in the “email” tube to send an email notifying the user that the post has been deleted.

- # Email Worker:
    - Retrieves the job from the “email” tube queue then processes the message.
    - Depending on the source of the message from the queue, “like” or “poll”, the body of the email is tailored to the offending resource.
    - Then the email is sent to the user.